LA-8849-MS

c. 2

A Philosophy of Supercomputing

University of California

LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos, New Mexico 87545

**UNITED STATES
DEPARTMENT OF ENERGY
CONTRACT W-7405-ENG. 36**

LA-8849-MS

UC-32
Issued: June 1981

A Philosophy of Supercomputing

Jack Worlton

A PHILOSOPHY OF SUPERCOMPUTING

by

Jack Worlton

ABSTRACT

This paper is the text of a lecture given at the Conference on High-Speed Computing, Glenenden Beach, Oregon, on March 30, 1981, sponsored by Los Alamos National Laboratory and Lawrence Livermore National Laboratory. It critically examines and attempts to clarify some of the commonly held beliefs about supercomputing, including basic problems, principles, and motivating concepts.

1. INTRODUCTION

The purpose of studies in the philosophy of technical subjects, such as the philosophy of science or, in this instance, the philosophy of supercomputing, is to critically examine and to attempt to clarify commonly held beliefs about the subject, including basic problems, methods, principles, and motivating concepts [1]. The classical role model for such a critical analysis is provided by the philosopher Socrates, who succeeded so well in critically examining the beliefs of his time that he was offered a special after-dinner drink. I trust I will not succeed that well. The views expressed here are personal views, not the views of the Los Alamos National Laboratory or the Lawrence Livermore National Laboratory (referred to herein as the laboratories), or the Department of Energy.

The year 1981 marks the 38th anniversary of the founding of the laboratory at Los Alamos, the 35th anniversary of the dedication of the first electronic computer (ENIAC), and the 29th anniversary of the founding of the laboratory at Livermore. In the years following these events, there has been a "symbiotic" relationship between science and computing, in which science has provided motivation and funding for the development of supercomputers, and supercomputers have provided means of solving problems in science that are otherwise intractable. In this span of time, the speed of computation has increased from a few operations per second in the electromechanical computers used at Los Alamos in the war years, to more than 10^7 operations per second in current Class VI supercomputers (see Fig. 1). This change of some seven orders of magnitude in less than four decades is unprecedented in the history of technology. Yet we meet here today to explain to the supercomputer designers that we have an urgent need for an increase of one to two orders of magnitude in computation speed in this decade. The effects of this projected speedup in computer capability would be to:

- Reduce the time needed to complete computationally dependent Department of Energy (DOE) programs. For example, a problem that now requires 100 hours of central-processor unit (CPU) time on a Class VI computer requires several weeks of real time to complete, because no single problem can be run to completion in one session, but must be run a few hours per night. A speedup of 100 to 1 would reduce the real time for problem completion from several weeks to just one night. Further, problems that now require 1 hour of CPU time on a Class VI computer must be run overnight, but these would be reduced to less than 1 minute and could be done interactively many times per day. Thus, the programmatic effect of the 100 to 1 reduction in compute time would be to reduce the time to complete a given project and thereby reduce the time to complete a given defense or energy program.

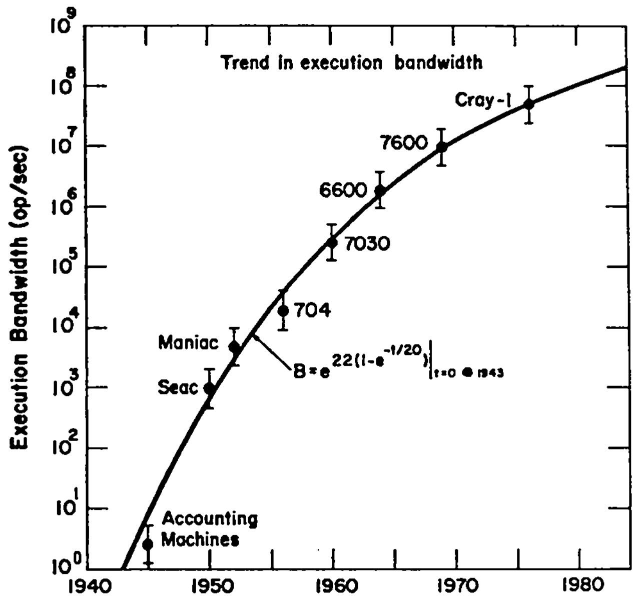


Fig. 1. Trend in execution bandwidth at Los Alamos.

- Increase the quality of the defence and energy products produced by DOE. The products with which DOE is concerned are radically different from ordinary consumer products because they include such things as the safety and security of nuclear weapons and nuclear reactors. The proposed 100 to 1 speedup in computer performance would allow DOE to explore more options in meeting its programmatic goals and thereby increase the quality of these products that are of such vital concern to the nation.

- Reduce the cost of achieving a given level of performance in a defense or energy system. It has been well said that time is money. Reducing the time needed to complete DOE programs would be one way to reduce the cost of these programs. DOE is concerned here not only with the cost of computing for a given project, but with the much larger cost of the project itself. For example, recent estimates are that the cost of the first demonstration power plant for magnetic fusion will be about $1 billion. Before such a plant is built, its total operational environment must be simulated to assure that the optimal configuration of the plant is used. This will require a supercomputer of greater computational capability than any now available.

- Increase the productivity of the most valuable resource the laboratories have--the scientists and engineers who work on defense and energy programs. As important as computers and other physical resources are, their productivity is of secondary concern to that of people. No computer has yet had an idea of how to make a safer nuclear weapon or a safer nuclear reactor, for example. Only people contribute in this way.

Although there is little doubt of the need for this increase in supercomputer capability, we would do well to bear in mind that improvements in supercomputing include more than just the speed of the computer hardware. There are many examples in which improvements in software and algorithms have contributed as much to large-scale scientific computing as have improvements in hardware. Indeed, with the slowing rate of advancement in the speed of semiconductor components, we are faced with an era in which improvements in supercomputing will have to come largely from advances in computer structures, software, and algorithms.

2. TECHNOLOGY TRENDS

Innovation in supercomputers is based in part on innovation in the components of which they are built and in part on innovation in the logical structure of the systems. In the past, many of the advances in supercomputers, such as the transition from the CDC 6600 to the CDC 7600, were based in large measure on faster components, but this option now seems inadequate by itself to meet our needs.

The major design changes in the Class VI computers are an indication of things to come. Components were not fast enough of themselves to achieve the required speedup, so the structure of the computers was changed to include explicit vector operations. This change in turn required the most far-reaching changes in code structure since the 1950s when floating point replaced fixed point.

The most often-quoted trend in component technology is "Moore's Law," which states that the number of components per chip will double every year [2]. This trend continued for a remarkable number of years. In 1979, however, Gordon Moore of Intel, the author of Moore's Law, noted that since 1975 the number of components per chip has been doubling not every year, but only every two years [3]. Leo Rideout of IBM also notes this slowing rate and comments that "Difficulties in designing circuits of much higher complexity could further diminish the growth rate in the 1980s" [4]. The delays encountered in the development of the 64K dynamic random access memory chips are symptomatic of this trend. The literature is full of the design difficulties and the high costs of developing VLSI components.

There are, of course, always positive announcements coming out of the components industry, and some of the developments made in connection with the Department of Defense's Very High Speed Integrated Circuit (VHSIC) program are especially noteworthy for advances that might be achieved by the middle and the latter half of this decade. gallium-arsenide and Josephson Junction technologies also hold promise for the latter part of the decade. But even the most optimistic views of component development lead us to the conclusion that components will not become faster by two orders of magnitude during the 1980s.

Thus, if we desire faster computers, we are led inevitably to achieving higher speeds in other ways--by changes in the structure of computers, by advances in software, by new physics models and algorithms, or by some combination of these options. All of these options have one thing in common--they require major commitments of manpower by the users.

The first option, changes in the structure of computers, is almost sure to affect supercomputer design in this decade in the form of explicit parallelism. This conclusion has some far-reaching consequences. It means, for example, that the major overhaul of application codes undertaken by the laboratories to take advantage of vector operations may be followed closely by another major overhaul to take advantage of explicit parallelism. The term "explicit parallelism" is used in contrast to the implicit parallelism that the industry has been using for more than two decades in designing faster supercomputers. Explicit parallelism requires the attention of the user. It cannot be ignored as could many of the implicit parallelism features of the past. Thus, an important part of understanding supercomputing for the 1980s is to understand parallel processors.

3. TAXONOMY AND NOTATION FOR PARALLEL PROCESSORS

The commonly used taxonomy of parallel processors is that proposed by M. Flynn in 1972 [5] and illustrated in Fig. 2. This taxonomy has been quite useful during the decade of the 1970s, but it has a serious flaw: the category of Multi-Instruction/Multi-Data (MIMD) includes not one but three separate architectures. To see these hidden architectures, consider Fig. 3, in which we classify computer architectures according to the relative number of execution units (E) and instruction units (I). With E less than I, we have a <u>shared resource</u> architecture, and a single instance of this architecture has a single execution unit serving many instruction units. With E equal to I, we have a <u>symmetric</u> architecture, which has provided the mainstream of computer design for some three decades. And with E greater than I, we have a <u>synchronized</u> architecture, in which a single instruction unit synchronizes the work of many execution units. In Fig. 4, we see the multiple forms of these basic architectures that clarify the three MIMD architectures. This taxonomy can be extended to a 12-way classification by classifying the E units according to the number of results that can be generated for each instruction issued: if the results/E-unit/instruction-issued equal 1, we have a scalar architecture; if the results/E-unit/instruction-issued can be greater than 1, we have a vector architecture.

Computing is sometimes called "computer science," but its lack of standard taxonomy and terminology is one of the marks of an immature science. Other, more mature sciences, are characterized by having systematic taxonomies of the objects with which they deal. On New Year's Day of 1730, the dean of the University of Upsala, Sweden, found on his desk a manuscript by an unknown student entitled, "Preliminaries on the Marriage of Plants" [6]. This manuscript was written by a student named Linnaeus, and it contained an idea for the classification of plants based on their reproductive methods that Linnaeus later used to classify all plants. Linnaeus also devised the binomial system of naming plants and animals using Greek and Latin words. On its way to becoming a science, computing has yet to find its Linnaeus to help it create a standard taxonomy.

Computing, however, does have some established nomenclature. Three items of standard notation for parallel processors commonly used are

T_p = the time to complete a task using p processors,

$S_p = T_1/T_p$ = speedup, and

$E_p = S_p/p$ = efficiency.

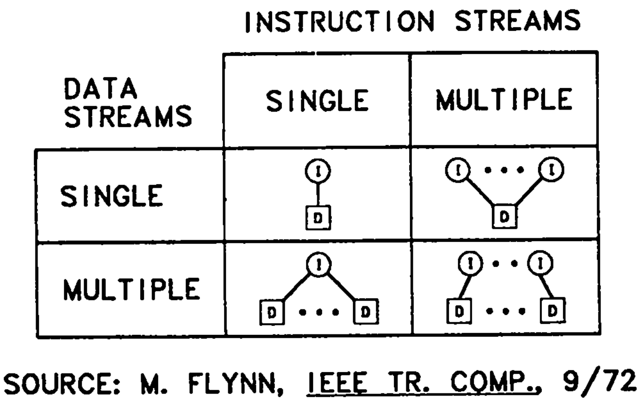
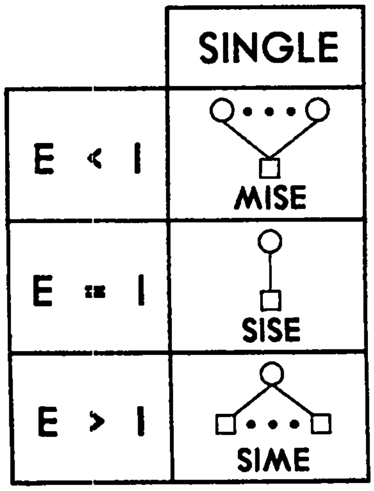


Fig. 2. Flynn's taxonomy of computer architecture.



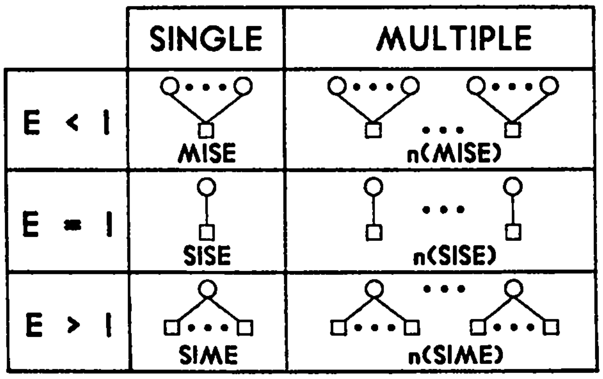


Fig. 3. A basic taxonomy of computer architecture.

Fig. 4. An extended taxonomy of computer architecture.

4. AN HISTORICAL PERSPECTIVE OF PARALLEL PROCESSING

One of the bits of folklore we often hear in discussions about supercomputers is that our roots lie in serial computation, that parallel computing is an unnatural way of computing, that humans just do not think this way, and therefore we will have almost insuperable difficulties making the transition to parallel computation. As is often true of folklore, there is some truth in this statement, but it ignores the frequent use of parallel processing by human beings. In fact, parallel processing is as natural as preparing a meal. Any cook who has several dishes to prepare, each of which takes a different amount of time to cook, uses parallel processing. The longest-cooking dish is started first, then the second-longest cooking dish, and so on, with everything coming out at the same time for the dinner. If a cook can do it, why can't a computer scientist? Automobile engines are marvels of parallel processing: the fuel and electrical systems must work in parallel, and within each of these systems there are many subsystems that work concurrently. If a mechanic can understand and work with parallel processing, why can't a computer scientist? And, all managers are familiar with parallel processing because they typically control many projects and many people working concurrently on each of these projects. If managers can work with parallel processing, why can't a computer scientist?

It might be argued that these are not calculating tasks, and that parallelism in calculation is indeed novel. But is it? In David Kuck's "Survey of Parallel Machine Organization and Programming" written for Computing Surveys in 1977 [7], he pointed out a number of historical precedents for parallel processing. For example, Babbage's Analytical Engine was described by Menabrea in 1842 as being able to prepare several results at once, although there is some question about whether this idea survived. Kuck's paper inspired me to think back on other, even earlier, uses of parallelism in computation; some examples follow.

The earliest instance of a parallel processor I can document is the so-called "Salamis tablet," found on the Greek island of Salamis, and dated to about the second century B.C. [8]. This device has three calculating positions. We are unsure how the device was used, but the conclusion is almost inescapable that the three calculating positions must have been used simultaneously, either for reliability through calculating the same result or for faster completion of calculating tasks that had been decomposed into parts.

The Europeans used the line abacus as their main means of calculation until about the 16th century A.D. [8], and the concept of parallel processing was used in these devices. Three of these calculating tables with line abaci incised into their surfaces can be seen in German museums. Each of these has more than one calculating position, as did the Salamis tablet. There are also line drawings dating from the 16th century that show several operators using these positions simultaneously.

We have already noted Babbage's design of the early 19th century. Later in that century, Herman Hollerith designed for the U.S. Bureau of the Census a tabulator that was quite successful, primarily because he tabulated all of the data on a card in parallel, using 40 tabulators that worked concurrently [9]. For example, if there were 10 items of information to be tabulated on a card, then all 10 of them would be counted at once. The processing elements are quite simple, being just counting devices, but the use of parallel processing is clear.

In the 1920s, A. J. Thompson connected four Trimphator calculators so that the output register of one calculator could provide input to the next to create a difference engine. He thereby created an interesting instance of a mechanical multiprocessor [10].

Punched-card accounting machinery was used for scientific computing at Los Alamos beginning in 1944, and this equipment was used in a parallel computing mode to shorten the time to complete the early weapons calculations. Richard Feynman, later a Nobel Laureate, devised parallel processing methods that increased the throughput by a factor of 9 [11].

Most people have forgotten that the first electronic computer, the ENIAC, was capable of parallel operations. It contained 20 adding-storage registers, a multiplier, and a divider/square-rooter. The control of operations in the ENIAC was implemented through a "master programmer" that could initiate several operations simultaneously. A description of the ENIAC published in 1945 states [12, p. 3-3]

> Since the ENIAC contains a number of trunk circuits, operations between various pairs of ENIAC units can be carried out simultaneously. This is possible not only because of this multiple trunk system, but because all units are synchronized by permanent electrical connection with the 'cycling unit.' Therefore if several operations are started simultaneously between various units of the ENIAC, and since all of these are timed from one and the same circuit, the various operations will end at known times relative to one another. Thus it is possible to plan the next group of simultaneous operations with the assurance that all of the prerequisite steps of the first group have been completed.

The authors of this report (Eckert, Mauchly, Goldstine, and Brainerd) discussed several levels of parallelism, including both single- and multiple-instruction streams. They noted the tradeoffs between the speed achievable with "multiple" (parallel) operation and the economy achievable with serial operation [12, pp. 4-3 to 4-5]. They concluded that serial operation was to be preferred in the post-ENIAC computers for two reasons: (1) new components were available that were fast enough so the desired speed of operation could be achieved without the complexity and the expense of building multiple processing units, and (2) programming for serial operations would be simpler than for multiple (parallel) operations. The fact that components are now no longer fast enough for our needs simply requires us to return to the problems faced by Eckert and Mauchly in designing the ENIAC. In designing parallel processors for the 1980s, the computing industry will not be starting onto a new path, but merely coming full circle onto an old one.

The descendants of ENIAC helped create the impression that computing is by its nature a serial activity. A few designs have incorporated explicit parallelism. Kuck mentions the Bell Labs Model V built by Stibitz and Williams in the late 1940s, a multioperation processor oriented around a drum memory by Leondes and Rubinoff in 1952, a drum-memory multiprocessor proposed by Konrad Zuse of Germany in 1958, and a number of proposed and real multiprocessors in the 1960s and 1970s [7]. Many vendors offer more than one processor in their product lines, including CDC, IBM, Univac, Honeywell, and Burroughs. However, uniprocessor designs have been the mainstream of computer architecture for some three decades.

The renewed interest in parallel processing is reminiscent of the philosopher Plato's theory of learning as anamnesis. Amnesia refers to forgetting, but anamnesis refers to a remembering of that which has been forgotten. Plato taught that we do not learn new things; we merely remember things we have forgotten. Whether this is true in general may be questioned, but for parallel processing Plato's point is well taken.

5. THE PROBLEMS OF PARALLEL PROCESSING

It is inherent in the nature of the future that it contains both problems and opportunities. In his book Future Shock Alvin Toffler argues that we can manage the future only to the extent that we can anticipate it [13]. Thus, it behooves us to try to anticipate both the problems and the opportunities of parallel processing.

To analyze the problems of parallel processing, I will use one of the fundamental principles of supercomputing, one that I call "Amdahl's Law." In 1967 Gene Amdahl presented a paper to the Spring Joint Computer Conference [14, 15] in which he warned that when we build computers with two distinct modes of operation, one high speed and another low speed, we thereby create a processor whose overall operation will be dominated not by the high-speed mode but by the low-speed mode, unless the fraction of results generated in the low-speed mode can be essentially eliminated. He further argued that eliminating this low-speed fraction would not be feasible in general-purpose computing. Amdahl's paper has been widely quoted and sometimes referred to as folklore, but the correctness of the basic premise is easy to establish.

The conceptual basis of Amdahl's Law is illustrated in Fig. 5. Here we assume that we have a relay team whose members are a tortoise and a hare. We first have the tortoise run the course of 100 units of distance; it covers the distance in time T, and we show its relative speed as unity. We then have the tortoise run only half of the distance, taking time T/2, and the hare--which we assume here is infinitely fast--run the last half of the course in zero time. We then pose the question, "How fast is the team as compared to the tortoise alone?" If we concentrate on the speeds of the two runners, unity and infinity, we might conclude that the

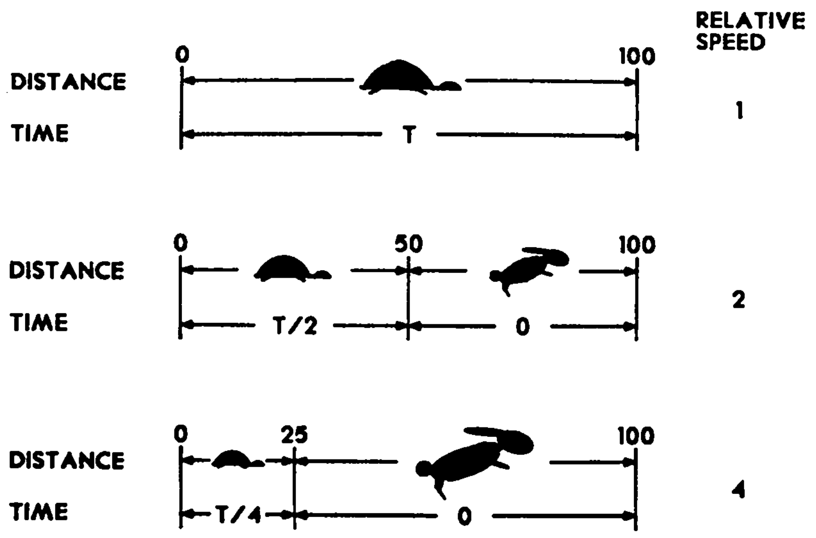


Fig. 5. The Amdahl relay team.

average speed is closer to infinity than to unity. However, if we note that the total time of the team was T/2, then it is obvious that the average speed has been increased by just a factor of 2. If we now have the tortoise run only one-quarter of the distance and the hare run three-quarters, we will increase the average speed of the team to just 4 times the speed of the tortoise alone. The point is just as Amdahl warned us: the slow member of the team dominates the overall performance.

We can express these ideas somewhat more formally by the following simple analytical model.

$$B = \frac{1}{F_H T_H + F_L T_L} ,$$

where

B = results generated per unit time,

F_H = the fraction of results generated in high-speed mode,

T_H = the time to generate a single result in high-speed mode,

F_L = the fraction of results generated in low-speed mode, and

T_L = the time to generate a single result in low-speed mode.

The validity of Amdahl's Law can be verified by using it to model existing supercomputers. We have done this for the Control Data Cyber 205 (see Appendix). Figure 6 compares the predictions of the model with benchmark data for the operation V = V + S (vector = vector + scalar), with contiguously stored vectors. The model also has been used to predict successfully the performance of the Cyber 205 for noncontiguously stored vectors and for more complex operations, including the triadic acceleration feature.

If we divide the numerator and denominator of Amdahl's law by T_L and let T_H go to zero to investigate the effect of infinitely fast high-speed mode, we have $B = B_L/F_L$, where $B_L = 1/T_L$ is the bandwidth in low-speed mode. That is, the speed of a computer having two modes of operation is limited by its low-speed mode divided by the fraction of results generated in that mode. For example, if half of the results are generated in low-speed mode, then the overall performance will be only a factor of 2 greater than if all results were generated in low-speed mode, even if the high-speed mode is infinitely fast.

It is also instructive to divide the numerator and denominator by T_H. We get

$$B = \frac{B_H}{F_H + F_L(T_L/T_H)} ,$$

where $B_H = 1/T_H$. This form of the model shows that the ratio (T_L/T_H) has the effect of increasing the fraction of results generated in low-speed mode. That is, the larger this ratio is, the greater is the effect of the low-speed mode, as shown in Fig. 7.

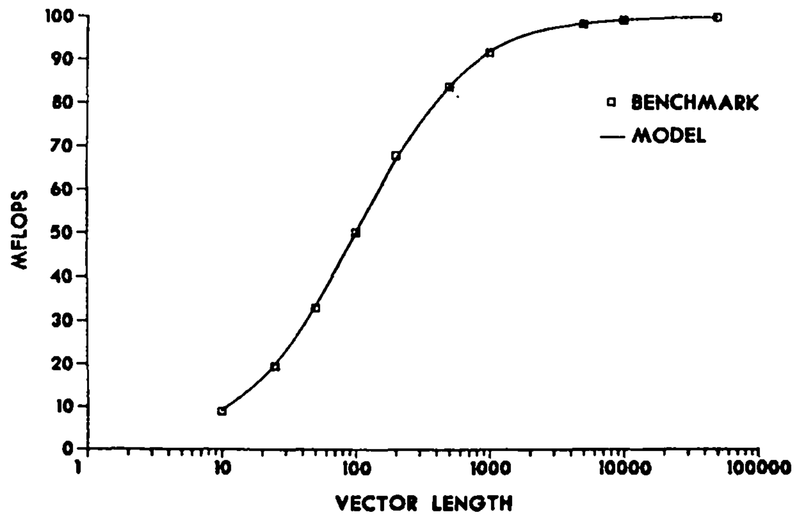


Fig. 6. Cyber-205 benchmark data vs analytic model.

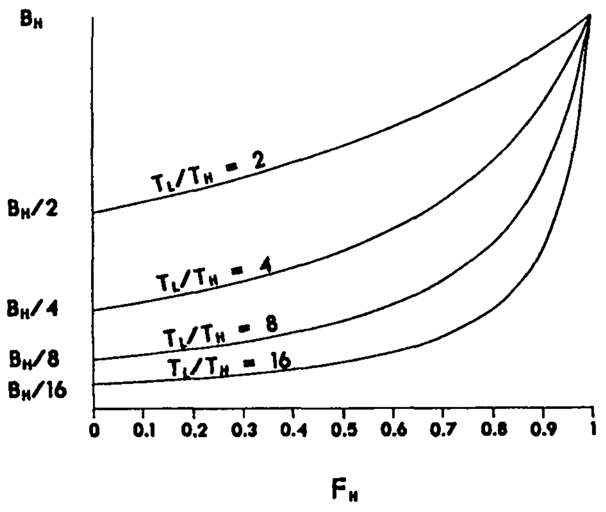


Fig. 7. Effect of T_L/T_H on performance.

This model can be used to explain why the first generation of vector processors was not, and why the second generation is, a marketing success. Figure 8 shows the performance curves of these two generations, compared to a reference computer, the CDC 7600. Here we idealize the first-generation vector processors to have a scalar performance of one-fifth of the reference computer and a vector performance of 5 times the reference computer (curve A). The performance curve remains below that of the reference computer until vectorization is about 83%; only with higher levels of vectorization is there an advantage to this type of vector computer as compared to the reference computer. This performance curve does not describe a useless computer, merely one that is special purpose; superior performance relative to the reference computer can be achieved only when this computer is applied to problems that can be vectorized above 83%. In contrast, the performance curve of second-generation vector processors is shown by curve B, where we assume the scalar performance to be a factor of 2 above the reference computer and the vector performance to be the same as in the first generation. This performance curve shows an advantage over the reference computer regardless of the vectorization level.

As the philosopher George Santayana remarked, those who will not learn from history are condemned to repeat it, and there is a danger that the errors of the past could be repeated with the technology of the future. For example, suppose a parallel processor is developed that has 16 processing elements (PEs), each of which is a factor of 2 slower than a reference Class VI computer (such as the Cyber 205 or the Cray-1), so the aggregate of 16 PEs operating concurrently would give a factor of 8 speedup over the reference computer. If we now assume that some tasks are done by just one PE and the rest are done using all 16 PEs, then the performance curve is as shown in Fig. 9. If a code for the reference uniprocessor were converted to run on one PE of the multiprocessor, it would run only half as fast as on the reference

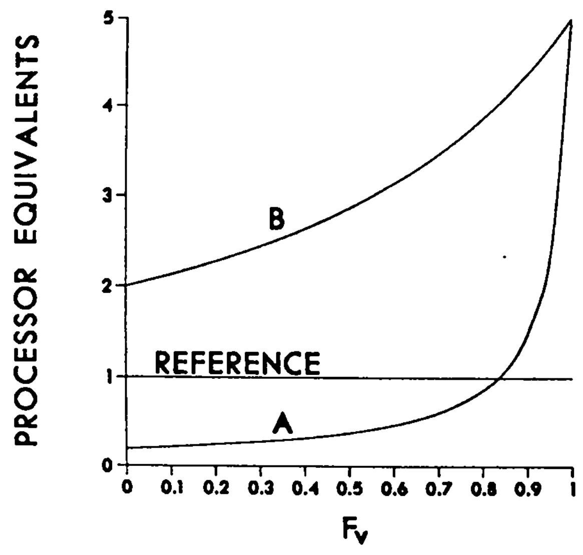


Fig. 8. Performance of first- and second-generation vector processors.

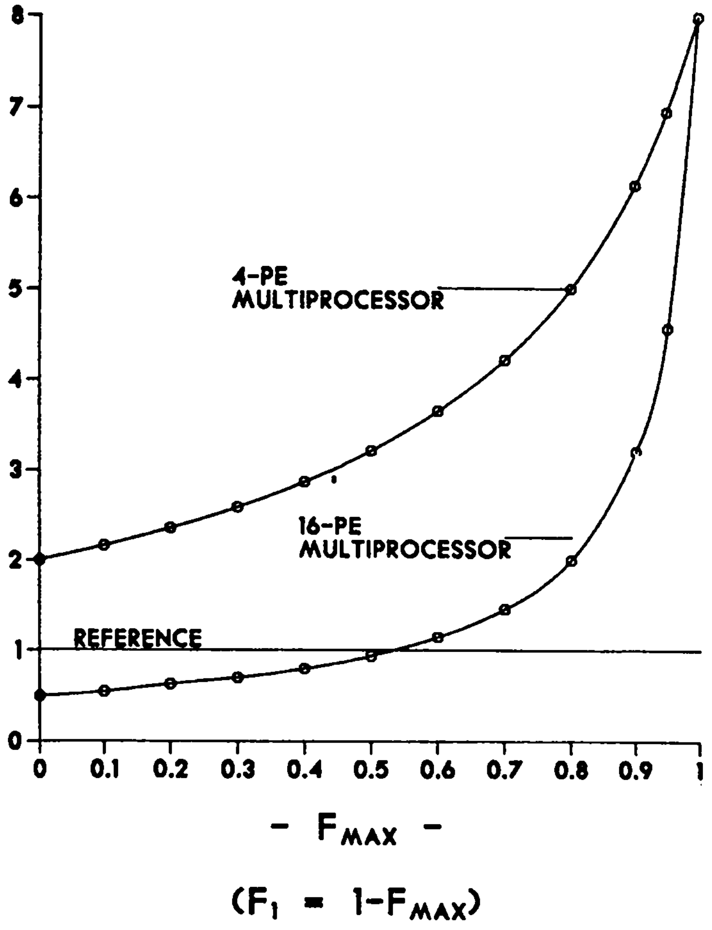


Fig. 9. Multiprocessor options.

computer. Even if the code were converted to use 16 PEs for some fraction of the tasks, no advantage over the reference computer would be achieved until that fraction exceeds 0.5. In contrast, suppose another option were to build a multiprocessor with only 4 PEs, each of which is faster than the reference computer by a factor of 2. Codes converted from the reference uniprocessor to the parallel processor would run no less than a factor of 2 faster, and at higher factors as the code is converted to multiprocessing operation.

There is a potential fallacy in this analysis, as pointed out by Kuck [16, p. 38]: we have assumed that some portion of the calculation, however small, would be executed in serial mode, that is, with only one PE. This is an assumption rather than a fact, and if it is false, then this analysis is invalid. For example, if the minimum number of active PEs were 4 in the 16-PE design, its performance would equal that of the 4-PE design having faster PEs. Optimizing compilers for parallel processors may help us avoid idle PEs, and thereby be an important aid in achieving this goal of essentially eliminating serial processing.

However, even a rather small amount of serial processing can significantly reduce the effectiveness of a multiprocessor, as shown by W. Ware in a paper in 1972 [17] and illustrated in Fig. 10. Here we assume that there is some small fraction of serial processing, k, and compute its effect on the speedup of a multiprocessor having 100 PEs. With no serial processing, the maximum speedup would be 100. However, with just 1% serial processing, the speedup would be reduced to about 50; with 2%

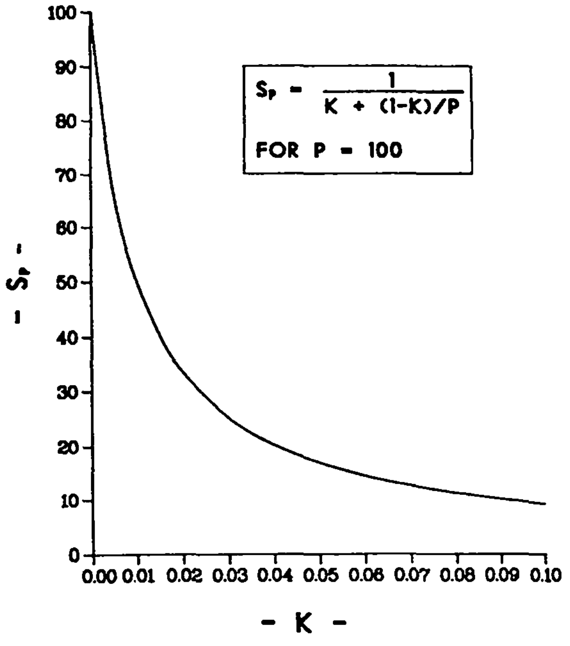


Fig. 10. Ware's model of multiprocessors.

serial processing, the speedup would be reduced to about 34; and with 4% serial processing, the speedup would be reduced to about 20. It is this reduction in potential benefits due to even very small amounts of serial processing that constitutes one of the problems in certain parallel processing designs and limits their usefulness to special-purpose computing.

This analysis of potential dangers of multiprocessing leads me to these conclusions.

- There is less risk in the use of multiprocessors having a small number of fast processors than there is in the use of multiprocessors having a large number of slow processors.

- Both vendors and users of multiprocessors should be strongly supporting compiler research for multiprocessors--the vendors to minimize marketing problems, and the users to minimize conversion problems.

6. THE OPPORTUNITIES OF PARALLEL PROCESSING

Even though there are problems in the use of multiprocessors, there are even greater problems in not using them; namely, we will not be able to obtain adequate computing capability for our programmatic needs. Only by using parallelism can we hope to achieve speed improvements of one to two orders of magnitude in this decade.

The question of just how much speedup multiprocessing will actually deliver has been a subject of speculation for some time. The most well-known speculation is probably that of M. Minsky [18], who conjectured in 1970 that the speedup due to multiprocessing would be limited to $\log_2$ of the number of processors used. For example, if Minsky's conjecture is correct, 1024 PEs would provide a speedup of only 10 over serial processing. Except for a small number of processors, this theory now appears to be overly pessimistic.

We can use a multiprocessor version of Amdahl's Law to analyze this potential:

$$S_p = \frac{T_1}{\sum_{i=1}^{p} f_i (T_1/i)} = \frac{1}{\sum_{i=1}^{p} f_i/i} ,$$

where f_i = the fraction of tasks that use i processors. The fractions, f_i, are problem dependent, and we can estimate the range of performance of a parallel processor by assigning these values to them:

- $f_1 = 1$. This defines a problem in which only serial processing is performed; this is the worst case of using a multiprocessor, because there is no speedup.

- $f_p = 1$. This defines a problem in which all PEs are active all of the time; this is the best case of using a multiprocessor, because speedup = p.

- $f_i = 0$ for all i except L and H, with L less than H. This is bimodal processing, one instance of which was analyzed in Section 5. For the general case, bimodal speedup = $1/(f_L/L + f_H/H)$. The effect of changing the modes of bimodal processing is shown in Fig. 11.

- $f_i = 1/p$ for all i. This is uniform multiprocessing, in which i processors are used for a fraction 1/p of the tasks.

R. Lee notes that $H_p = 1 + 1/2 + 1/3 + \ldots + 1/p$ is the pth harmonic number; that $H_p = \ell n\ p + .57721\ldots + O(1) = \ell n\ (1.78p)$; and that H_p goes to $\ell n\ p$ as p goes beyond bound [19]. Thus, for uniform multiprocessing, speedup is given by

$$S_p = \frac{T_1}{\sum_{i=1}^{p} (1/p)(T_1/i)} = \frac{p}{\sum_{i=1}^{p} (1/i)}$$

$$\cong p/(\ell n\ 1.78p) \longrightarrow p/\ell n\ p \quad .$$

These categories are summarized in Table I.

TABLE I

SOME GENERAL MULTIPROCESSING CATEGORIES

f_i	S_p	Description
$f_1 = 1$	1	Uniprocessing
$f_p = 1$	p	Maximum multiprocessing
$f_L = 1-f_H$	$1/(f_H/H + f_L/L)$	Bimodal multiprocessing
$f_i = 1/p$	$1/(\ell n\ 1.78p)$	Uniform multiprocessing

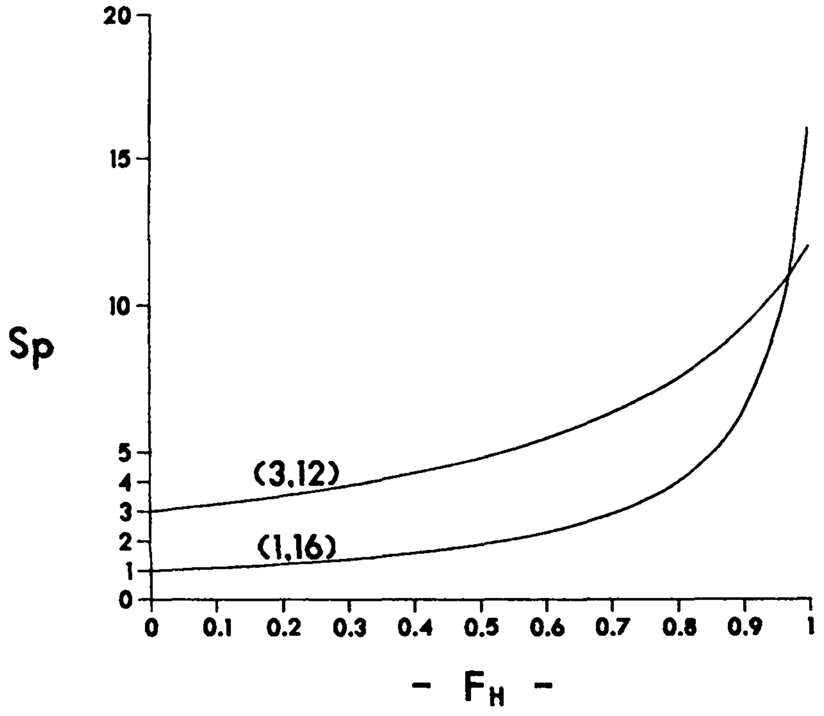


Fig. 11. Bimodal multiprocessing.

In work reported in 1972, Kuck concluded that, for a set of ordinary Fortran programs he studied, it was possible to achieve a speedup of more than 10 using only 32 processors, and that larger programs offered opportunities for even higher efficiency [16]. These data are plotted in Fig. 12. If it is indeed possible to achieve an efficiency of at least 30% for the large-scale application codes used at the laboratories, then multiprocessors with a large number of PEs would be attractive.

Figure 13 summarizes several performance projections for parallel processors as a function of the number of processors used. For purposes of illustration, the scale for the number of processors has been extended to 1000. However, a given calculation may reach its limit of parallelism before this number is reached, in which case the speedup would not increase for a larger number of processors.

Whether multiprocessors will be effective for the work of the laboratories is, of course, problem dependent, and the overall benefits are still unknown. Some problem categories now show promise, as will be reported by other speakers [20]. However, in making these evaluations of the effectiveness of multiprocessing, we must bear in mind Amdahl's original warning. It is not just an algorithm or a set of algorithms that we must evaluate, but the total problem environment, including what Amdahl calls the "data management housekeeping."

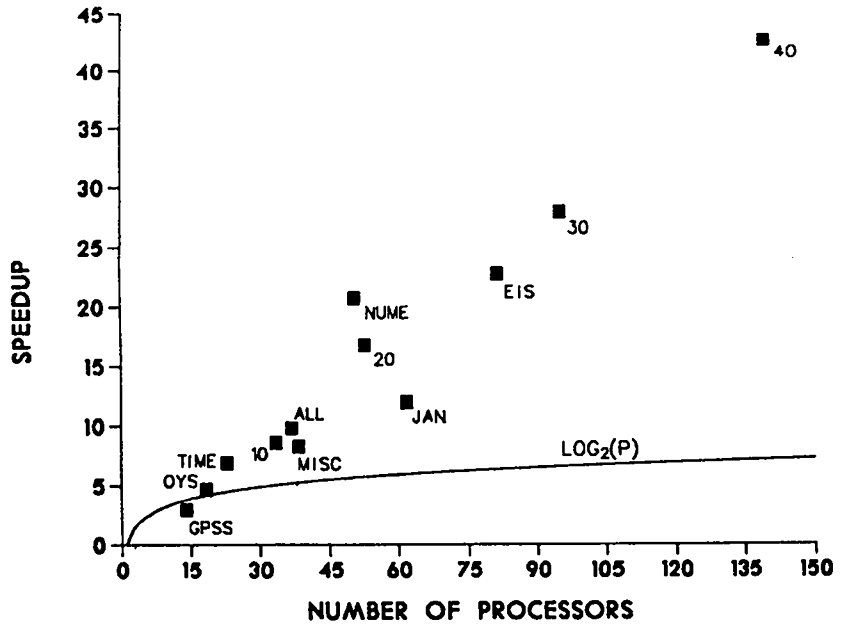


Fig. 12. Kuck's speedup analysis for ordinary Fortran programs.

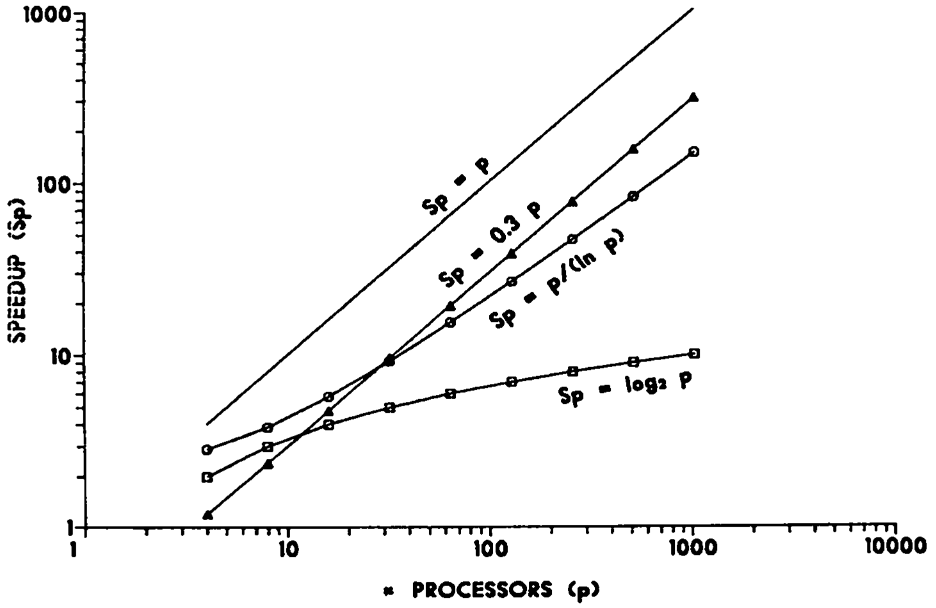


Fig. 13. Comparison of several speedup projections.

7. COMPUTATIONAL PHYSICS: THE SOFTWARE PROBLEM

There are frequent references in the literature to the "software problem"--to problems such as the high variability in the productivity of programmers, the time and cost overruns in the production of software, and the unreliable nature of software. Programmers often discuss the software problem in terms of GO-TO, IF-THEN-ELSE, modularity, and in general what is called structured programming. All of the benefits of structured programming are necessary for the solution of the software problem, but they are not sufficient. Software problems afflict computational physics quantitatively through duplication of effort, and qualitatively through the inadequate languages computational physicists are forced to use.

Duplication of effort in software development occurs primarily because so much software is not reusable. In this context it is helpful to distinguish between ad hoc software and generic software. Ad hoc software is software that is developed for a special purpose, software that is not well documented or tested, software whose existence is not well publicized outside the organization that produced it. It is "throwaway" software [21]. Because of these characteristics, this kind of software is a major cause of low programmer productivity. Even if it were produced with careful attention to the best practices of structured programming, it would still cause low programmer productivity because the same code would be produced repeatedly. Software is often deliberately not publicized to outside parties because of the time and cost needed to prepare it for use by others and because of the worry about external demands on the time of the developer. These are valid concerns, but the long-term effect is to lower the productivity of all programmers.

Generic software, on the other hand, is characterized by (1) the intention to solve some problem so well that it need not be done over again, (2) the excellence of the programming that produced it, (3) the documentation of its functions, usage information, and source code, (4) the testing and certification of the code, and (5) the publication of this information so that others can use it with a minimum of effort. Examples of generic software are the EISPACK [22] and the LINPACK [23] routines. For users of eigensystems and linear systems, the excellence of these routines makes it unnecessary to program them again. They constitute a solved problem in software.

Computer science, in general, and computational physics, in particular, are characterized by a relatively small body of generic software and a relatively large body of ad hoc software. This is to be contrasted with science and engineering, in which there is a large body of generic knowledge, with ad hoc investigations aimed at becoming generic knowledge. In part, this contrast between computer science and most other fields of science is the result of the longer time that scientists and engineers have had to produce generic knowledge as compared to computer scientists. However, it exists also because programmers are not disciplined to develop generic software. They create programs to meet their own limited needs rather than considering the needs of the larger community of users.

Some programs belong in the ad hoc category because we do not yet know how to do them adequately. Many of the codes used at Los Alamos and Livermore are in this category because the laboratories work at the frontiers of science and they are exploring new fields. It would be premature to package these codes, because they do not represent solved problems. However, even here it is possible to move some portions of these codes into the generic category. Many of the major production codes at the laboratories are already decomposed into generic code modules of modest size. For areas that are not changing, these generic modules could be used as building blocks for the development of future codes, not only in the organization that produced them but for others as well. The documentation, certification, and publication of these modules would mean that each laboratory could take better advantage of work at the other laboratory and thereby reduce duplication of effort.

An additional advantage of this approach to code development would be that the generic modules might be usable for parallel processing because decomposition of major codes into smaller, independent, generic modules is an important part of converting codes from uniprocessing to multiprocessing.

Another task that needs to be done to solve the software problems of computational physics is the development of a high-level language for the use of physicists. Why is it that it often takes 50 000 to 100 000 lines of Fortran to communicate a code specification to a computer? In part it is because of the vast gap between the knowledge base of a scientist and the knowledge base of a computer. One indicator of that gap is vocabulary. There are more than half a million words in the English language, and a Ph.D. physicist has a recognition vocabulary of some 80 000 to 100 000 words. By contrast, a child entering school may have a vocabulary of only 3000 to 4000 words [24]. A computer system, including the hardware and system software, has a recognition vocabulary that I estimate to be on the order of 2000 words. Even if we assume that this estimate is off by a factor of 4 to 5, this still leaves a computer with a vocabulary that is an order of magnitude smaller than that of a physicist. The image that comes to mind is one suggested by James George--Albert Einstein trying to communicate relativity to a child using pidgin English.* One of the reasons why physicists need code developers is that they cannot communicate directly with a computer at their own level. In a religious context this kind of person is a shaman or priest, someone who can communicate with some mysterious and powerful entity that is impossible for ordinary people to talk to.

Oliver McBryan of New York University, a speaker at a recent workshop on computer modeling at Los Alamos, made the following point [25].

*Personal communication, James George, Los Alamos National Laboratory (1981).

> The present almost universally awful operating systems and languages have repelled most outstanding physicists and have helped to give the whole area of computational physics a bad name... . A computer language for the use of theoretical physicists should be considerably more powerful than Fortran.

Our commitment to Fortran is almost total, because we have such a large set of working codes in that language, not because of the excellence of the language. Often the attitude is "if it works, leave it alone; if it is fast, it is outstanding" and "better is the enemy of good."

The problem this conservatism causes is well illustrated by the following parable. A lumberjack was hired at a lumber camp, and the first week he cut down more trees than anyone else in the camp, so the foreman was well pleased with his work. The second week, however, his output dropped to the same level as the other lumberjacks, and the third week, his output dropped even lower. The foreman called him in to warn him that unless he increased the number of trees he cut down, he would be fired. The lumberjack said he couldn't understand it, that he was working as hard the third week as he was the first week. The foreman then asked him how long it had been since he had sharpened his axe, to which the lumberjack indignantly replied, "Sharpen my axe! I'm too busy cutting down trees to sharpen my axe!"

Just so, by taking the view that we cannot afford to change the way we do things, we run the risk of low productivity--not only for programmers but more importantly for the scientists and engineers on whose work the success of the laboratories depends.

We must get beyond programming languages to languages that allow scientists and engineers to describe the results desired, rather than merely how to carry out the steps of the computation [26]. A better language than Fortran need not mean abandoning Fortran; a high-level language can be used in conjunction with Fortran. Only when we have such a high-level language will we really solve the software problem for computational physics. We must stop forcing physicists to speak Fortran and teach computers to understand physics and mathematics.

In summary, to solve the software problems of computational physics at the laboratories, we must

- decompose our major codes into modules of generic value, so they can be used as building blocks for future use;
- document, certify, and publicize these modules, so that their existence and characteristics are known and they can be used with a minimum of effort;

- form an interlaboratory library of generic application modules; and
- develop a high-level language for the use of physicists that allows them to deal more directly and simply with computing resources.

8. BALANCING THE SUPERCOMPUTING BUDGET

The problems with software for computational physics raise a question about whether we have our budgets for supercomputing in the proper balance. As noted earlier, there are many instances in which improvements in software and algorithms have achieved improvements in supercomputing that match improvements in hardware. To cite a recent example, Thomas Jordan of Los Alamos found a way to vectorize an equation-of-state interpolation routine that previously was thought to be not vectorizable [27]. He thereby reduced the running time by a factor of 3 to 4 over the original code, and in that one action he advanced the state-of-the-art for that area by a supercomputer generation. Unfortunately, there are too few people with these skills. We have an urgent need to hire and train more people to help us achieve similar gains in other areas.

The problem of inadequate support for code development and algorithm research was strongly emphasized in the recommendations of a code review panel at Los Alamos published in January 1981 [28]. Some examples follow.

> Fundamental advances in the capabilities and efficiency of the plasma simulation codes will require basic research in numerical methods. This is perhaps our weakest area at present; adequate manpower is simply not available for investment in long-range studies.
>
> ...the process of winnowing, implementing, and testing new algorithms is quite time consuming.... If we wish to pursue numerical research, some augmentation of manpower is essential.

No computer is so powerful that it can carry the burden of inadequate software and algorithms. In the jargon of the street, the word "user" refers to a person who is addicted to some drug, and "speed" refers to one of these addicting drugs. The application of these terms to supercomputer users is all too clear: we are addicted to supercomputer speed, and we periodically need a fix of this drug or we have withdrawal symptoms and go to our pushers to convince them to give us another shot. Computational physics might be healthier in the long term if it were to invest additional resources in algorithm and software improvements, even if this investment requires limitations on hardware budgets.

9. LIMITS TO SUPERCOMPUTING SPEED

The growth curve in Fig. 1 that traces the history of the increase in computing speed approaches a limit of e^{22}, or about 3.6 x 10^9. This is merely an empirical curve, and it might be unrelated to future trends. However, the fact that trends in component technology are exhibiting the same shape is an indication that there may be underlying causes for this growth curve that will limit the speed that can be achieved in future supercomputers. Some of the barriers to such increases are discussed below and in Refs. 30 and 31.

- As binary elements become smaller, they become subject to spontaneous switching of their state; the spontaneous switching time depends exponentially on the size of the element, so binary elements cannot be made arbitrarily small.

- For a bistable element to maintain its state after it has been switched, it must be able to dissipate the energy that enabled it to surmount the potential barrier. Otherwise, this energy might be absorbed by a neighboring element, which would then undergo an unwanted change.

- Logic elements must transfer their heat to a larger solid body, which in turn typically transfers it to a liquid that can carry it away. The temperature rise experienced by a logic element when it switches state is inversely proportional to the square of the diameter of the element, so this is another reason that the elements cannot be made arbitrarily small.

- As devices get smaller, uniformity in fabrication becomes increasingly difficult to achieve; a flaw that was unimportant in a large element may become decisive in a small element. Fabrication will become an increasingly difficult problem in achieving smaller dimensions.

A question of immediate concern is whether these and other limits will prevent the achievement of the speeds needed in the next decade. This does not now seem likely; in fact, it is generally believed that fundamental limits will not be encountered in this century [29-31].

Some of the factors that will offer opportunities for significant growth of performance in this decade include:

- Component costs. Component costs have been declining by about 18% per year. If this decline were to continue through this decade, designers would be able to use four to five times as many components in a supercomputer as are currently used, at constant cost. Lower component costs will therefore enhance the benefits of parallel processing.

- Faster silicon technology. The current line widths of about 2.5 μm in silicon technology are now thought to be reducible by about a factor of 10 by the year 2000 [30]. The scaling laws of silicon technology reduce both switching time and transmission delays as line widths are reduced in size and as more circuits are fabricated on each chip. However, resistance increases as line widths are reduced, and it may be necessary to use lower temperatures to counteract this effect. Although these improvements alone will not yield the speed increases required in this decade, they will make an important contribution.

- Parallelism. Parallelism, combined with component cost decreases and improved silicon component technologies, seems capable of providing performance improvements in general-purpose supercomputers of factors of 5 to 10 by the middle of the decade, and even greater improvements by the latter half of the decade.

- New circuit technologies. The contenders here are gallium arsenide and Josephson Junction (JJ) technologies, with JJ apparently offering opportunities for greater advances. By about 1990, the JJ technology appears capable of producing a computer that is about a factor of 10 faster than the current supercomputers, using current architectures. The combination of JJ and parallel architecture offers hope of more than a factor of 10 increase in the speed of supercomputers.

The point here is that there are no barriers in technology to achieving increases of one to two orders of magnitude in the speed of operation of supercomputers. The only question is, "How soon can these increases be achieved?" Combining the potential advances of parallelism, lower costs, and new circuit technologies indicates that speeds of several billion operations per second should be achievable in the long term.

10. SHOULD THE LABORATORIES BUILD THEIR OWN SUPERCOMPUTERS?

The laboratories are in the defense and energy research business, not the computing business, so it would be a major change of role for them to develop supercomputers. In the early history of electronic computing, many laboratories and universities developed their own computers because there were no adequate commercial offerings. This question is relevant because, again, we feel strongly that commercial offerings are inadequate. The very holding of this conference is a strong message to the vendors that their current offerings are inadequate for our needs. Also, the development of the S-1 computer at Livermore shows that the needs we are discussing here are taken very seriously. There is some feeling at the laboratories that the supercomputer vendors are driven more by marketing considerations than by the desire to achieve the highest possible performance. An orderly factor of 4 improvement in each supercomputer generation may fit the vendors' marketing needs, but an orderly factor of 10 would fit the laboratories' needs much better.

In any discussion about building our own supercomputers, we need to bear in mind the hazards inherent in the attempt to build a supercomputer, hazards the supercomputer vendors live with routinely, including

- a nonzero probability that the computer will never be completed at all;
- even if it is completed, the time for completion may be much longer than predicted;
- even if it should be completed reasonably on time, the costs may be much higher than expected;
- even if time and costs are in line with expectations, the performance may be disappointing; and
- even if all else goes well, the reliability and maintainability of the computer may not be satisfactory.

It is our hope that we will not find it necessary to develop our own supercomputers, but the probability is not zero. The laboratories may be forced into this role because of the lack of adequate products from the vendors.

American computer vendors have had a monopoly on this market for about 20 years, but the announced intentions of the Japanese computer manufacturers to build supercomputers [32, 33] may require a change in the development and marketing plans of American supercomputer vendors. It would be highly regrettable for the American supercomputer business to suffer the impact from foreign competition that has been experienced by the American steel, stereo, and automotive industries.

11. CONCLUSIONS

(1) Increases of one to two orders of magnitude in the operating speed of supercomputers cannot be attained in this decade without changing the logical structure of these computers. This in turn will require major commitments of manpower by users to adapt software and algorithms to the new structures.

(2) There is less risk in the use of multiprocessors having a small number of fast processors than in the use of multiprocessors having a large number of slow processors.

(3) Both the vendors and the users of supercomputers should be strongly supporting compiler research for multiprocessors--the vendors to minimize marketing problems, and the users to minimize conversion problems.

(4) The laboratories should decompose their major codes into documented, certified, and publicized modules of generic value, so that these modules can be used as building blocks for future use, including parallel processing.

(5) The laboratories should support the development of a language for computational physics that is much more powerful than Fortran. Fortran should not be the only language of computational physics.

(6) The laboratories should place increased emphasis on improvements in software and algorithms, even at the expense of lower budgets for hardware.

Progress in technology typically occurs when someone has a vision of how to create new tools and techniques for the solution of practical problems. It is the sweep and the clarity of this vision that carries others along and leads to action. Action in turn leads to a better understanding of the improved technology and the nature of the problems to be solved. Finally, this better understanding alters our vision of what can and should be done. Vision, action, and understanding thus form a feedback loop through which we pass repeatedly in the development of new and improved technologies [34].

The computing industry has benefited from the visions and the actions of many people, including Thomas J. Watson, Sr., of IBM, whose vision helped create the data-processing industry; William Norris and other founders of CDC, whose vision created a new source of large-scale computers for the using community; Seymour Cray, whose vision spans supercomputer design from cooling to components to architecture; and Gene Amdahl, whose vision created a new segment of the computing industry. That we are able to meet and discuss improvements to supercomputers is due in large part to these and others who had the courage to risk their finances and careers in bringing their visions into being. In expressing our vision of better supercomputers to the computing industry, we also need to express our appreciation to them for what they have already achieved.

The laboratories need to formulate their own vision of the future of supercomputing, including

- a clear understanding of the problems and opportunities of parallel processing,
- the need to support research <u>now</u> to prepare for parallel processing,
- the need to develop generic software for computational physics,

- the need to provide more powerful languages for computational physics, and

- the need to provide support for algorithm and software improvements commensurate with our support for hardware improvements.

We need to invent the future, not merely forecast it. We need to visualize the future in terms of what could be done, and then make it happen.

APPENDIX

AN ANALYTIC MODEL OF THE CYBER 205

Given Amdahl's Law, we expand the time to generate a result in high-speed mode to reflect the vector result-generation time of the Cyber 205, as follows.

Let t_v	=	the vector result-generation time for 64-bit operands using a single pipe,
n	=	the number of vector pipes,
x	=	the fraction of results generated in triadic mode, and
y	=	the fraction of results generated in 32-bit mode.

Then $t_v/[n(1+x)(1+y)]$ is the time to generate a result using n pipes, with a fraction x of triadic operations, and a fraction y of 32-bit operands.

Let t_g	=	the gather/scatter time, and
G	=	the number of gather/scatter operations per vector operation.

Then $t_g G$ is the time that must be added to the vector result-generation time to account for noncontinguous operands.

Let S	=	the vector startup time, and
L	=	the vector length.

Then $(S/L)(1+G)$ is the time that must be added to the vector result-generation time to account for startup of vector and gather operations. We assume here that these startup times are equal and constant.

Let t_s = the time to generate a scalar result,

F_v = the fraction of results generated in vector mode,

$1-F_v$ = the fraction of results generated in scalar mode, and

z = the fraction of scalar operations overlapped with vector operations.

Then $(1-F_v)t_s(1-z)$ is the time per result required to generate scalar operations.

Combining these timing contributions, and inserting them into Amdahl's Law, we have

$$B = \frac{1}{F_v\left[(S/L)(1+G) + t_g G + \dfrac{t_v}{n(1+x)(1+y)}\right] + (1-F_v)t_s(1-z)} .$$

Some standard values for the model parameters are as follows.

S = 1.0 μs (varies from 0.8 to 1.0 μs);

L = problem dependent (typically varies from 10 to 10 000);

G = problem dependent (0 for contiguous data, 1 if a single operand-vector must be gathered/scattered per vector operation, 2 if two operand-vectors must be gathered/scattered, etc.);

t_g = 0.030 μs;

t_v = 0.020 μs;

t_s = 0.225 μs;

n = 1, 2 or 4;

x = problem dependent (1 if all operations are triadic);

y = problem dependent (1 if all operands are 32-bit); and

z = problem dependent (1 if all scalar operations are overlapped with vector operations).

These are preliminary values subject to revision; they are based on a limited set of benchmarks run by Los Alamos National Laboratory [35] and on data published by Control Data Corporation [36].

REFERENCES

1. Ernest Nagel, "The Nature and Aim of Science," in Philosophy of Science Today, Sidney Morganbesser, Ed. (Basic Books, New York, 1967), pp. 3-13.

2. Robert N. Noyce, "Microelectronics," Scientific American, 237 No. 3, p. 65 (September 1977).

3. Gordon Moore, "VLSI: Some Fundamental Challenges," IEEE Spectrum 16, No. 4, 30-37 (April 1979).

4. V. Leo Rideout, "Limits to Improvement of Silicon Integrated Circuits," Digest of Papers, IEEE Compcon (San Francisco, CA, Spring 1980) pp. 2-6.

5. M. Flynn, "Some computer organizations and their effectiveness," IEEE Trans. on Computers, C-21, No. 9, 948-960 (September 1972).

6. D. D. Runes, "Carolus Linnaeus," A Treasury of Philosophy II (Grolier, Inc., New York, 1955), pp. 717-719.

7. D. J. Kuck, "A Survey of Parallel Machine Organization and Programming," ACM Computing Surveys, 9, No. 1, 29-59 (March 1977).

8. Karl Menninger, Number Words and Number Symbols (MIT Press, Cambridge, Mass., 1969) pp. 299-303.

9. Jack Worlton, "Pre-Electronic Aids to Digital Computation," in Computers and Their Role in the Physical Sciences, S. Fernbach and J. Taub, Eds. (Gordon and Breach, New York, 1970), pp. 40-41.

10. Charles Eames and Ray Eames, A Computing Perspective (Harvard University Press, Cambridge, Mass., 1973), pp. 94-95.

11. Richard P. Feynman, "Los Alamos From Below," in Engineering and Science, January-February 1976, pp. 25-27.

12. J. P. Eckert, Jr., J. W. Mauchly, H. H. Goldstine, and J. G. Brainerd, "Description of the ENIAC and Comments on Electronic Digital Computing Machines," AMP Report 171.2R (November 30, 1945).

13. Alvin Toffler, Future Shock (Bantam Books, New York, 1970), p. 379.

14. Gene Amdahl, "The Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," AFIPS Conf. Proc., 30 (1967) pp. 483-485.

15. Gene Amdahl, "Advances in Computer Technology," Future Systems Vol. 2, Infotech International, Ltd. (Nicholson House, Maidenhead, Berkshire, England, 1977), pp. 2-25.

16. D. J. Kuck, "Multioperation Machine Computational Complexity," in Complexity of Sequential and Parallel Numerical Algorithms, J. F. Traub, Ed. (Academic Press, New York, 1973), pp. 17-47.

17. Willis H. Ware, "The Ultimate Computer," IEEE Spectrum, 9, No. 3, 84-91 (March 1972).

18. M. Minsky, "Form and Content in Computer Science," ACM Turing Lecture, Journal of the ACM, 17 No. 2, 197-215 (February 1970).

19. R. Lee, "Performance Bounds in Parallel Processor Organizations," in High Speed Computer and Algorithm Organization, D. Kuck, D. Lawrie, and A. Sameh, Eds. (Academic Press, New York, 1977), pp. 453-455.

20. B. Buzbee and G. Michael, Abstracts of the Conference on High Speed Computing (March 31 to April 2, 1981, Glenenden Beach, Oregon).

21. Peter J. Denning, "Throwaway Programs," Commun ACM 24, No. 2, 57-58 (February 1981).

22. B. Smith, J. Boyle, B. Garbow, Y. Ikebe, V. Kleme, and C. Moler, Matrix Eigensystem Routines--EISPACK Guide, Lecture Notes in Comput Sci, Vol. 6 (Springer-Verlag, New York, 1976).

23. J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, LINPACK Users' Guide (SIAM, Philadelphia, 1979).

24. "Vocabulary," The World Book Encyclopedia, 20, (Field Enterprises Educational Corp., Chicago, 1972), p. 337.

25. Oliver McBryan, "Operating System and Software Support," notes for a presentation to the Computer Modeling Working Group, Los Alamos National Laboratory, January 21, 1981.

26. Terry Winograd, "Beyond Programming Languages," Commun ACM 22, No. 7, 391-401 (July 1979).

27. B. L. Buzbee, T. L. Jordan, and Jack Worlton, "The Challenge of Vector Computers," Los Alamos National Laboratory report (to be published).

28. H. H. Rogers, "Panel B Code Review Summary," Internal Los Alamos National Laboratory memo, January 7, 1981.

29. M. J. Freiser and P. M. Marcus, "A Survey of Some Physical Limitations on Computer Elements," IEEE Trans on Mag MAG-5, No. 2, 89-90 (June 1969).

30. Robert W. Keyes, "Physical Problems and Limits in Computer Logic," IEEE Spectrum, 6, No. 5, 36-45 (May 1969).

31. James D. Meindl, Electronic Engineering Times, Monday, March 2, 1981, p. 16.

32. Gina Bari Kolata, "Who Will Build the Next Supercomputer?" Sci 211, 268-269 (16 January 1981).

33. Japan Information Processing Development Center (JIPDEC) Report, Tokyo, Japan, Summer 1980.

34. Peter F. Drucker, The Age of Discontinuity (Harper & Row, New York, 1968).

35. Ann Hayes, "Preliminary Report on Cyber 205 Benchmark," Los Alamos National Laboratory internal memorandum, February 18, 1981.

36. Michael J. Kascic, Jr., "Vector Processing: Problem or Opportunity?" Digest of Papers,IEEE Compcon (San Francisco, Spring 1980), pp. 270-276.

Printed in the United States of America
Available from
National Technical Information Service
US Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Microfiche $3.50 (A01)

Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code
001-025	$ 5.00	A02	151-175	$11.00	A08	301-325	$17.00	A14	451-475	$23.00	A20
026-050	6.00	A03	176-200	12.00	A09	326-350	18.00	A15	476-500	24.00	A21
051-075	7.00	A04	201-225	13.00	A10	351-375	19.00	A16	501-525	25.00	A22
076-100	8.00	A05	226-250	14.00	A11	376-400	20.00	A17	526-550	26.00	A23
101-125	9.00	A06	251-275	15.00	A12	401-425	21.00	A18	551-575	27.00	A24
126-150	10.00	A07	276-300	16.00	A13	426-450	22.00	A19	576-600	28.00	A25
									601-up	†	A99

†Add $1.00 for each additional 25-page increment or portion thereof from 601 pages up.